(12) UK Patent Application (19) GB (11) 2 032 149 A

(21) Application No 7930110
(22) Date of filing 30 Aug 1979
(23) Claims filed 30 Aug 1979
(30) Priority data
(31) 53/110303
(32) 8 Sep 1978
(33) Japan (JP)
(43) Application published
30 Apr 1980
(51) INT CL³
G06F 11/20
(52) Domestic classification
G4A 12T ES
(56) Documents cited
GB 1507428
GB 1412246
GB 1355295
GB 1184160
GB 1181182
GB 1168414
GB 1163859
US 3517171 A
(58) Field of search
G4A
(71) Applicant
Fujitsu Limited, 1015,
Kamikodanaka,
Nakahara-ku, Kawasaki-
shi Kanagawa 211, Japan
(72) Inventors
Masamichi Iwama,
Satoru Tsushima,
Koichi Watanabe
(74) Agent
Marks & Clerk

(54) Transferring from Working to Standby Processor on Fault

(57) A multi-processor system includes at least a first processor and a second processor one of which performs data processing of successive processing steps. The processing processor transfers result data to the other processor each time a data processing step is completed, and the latter processor stores the transferred resultant data. Programs are segmented by respective predetermined breakpoints for a job. When the processing processor is in an abnormal condition, the other processor takes over the job from the nearest breakpoint by utilizing the last transferred result data. The arrangement requires a simpler interface than a fully duplicated redundently operated system whilst providing rapid transfer when a fault occurs.

GB 2 032 149A

Fig. 1

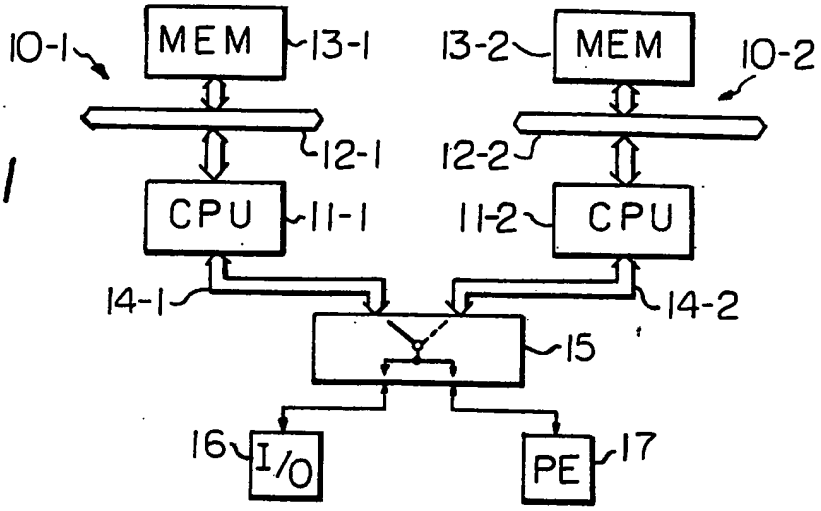

10-1
MEM
13-1
13-2
MEM
10-2
12-1
12-2
CPU
11-1
11-2
CPU
14-1
14-2
15
16
I/O
PE
17

Fig. 2

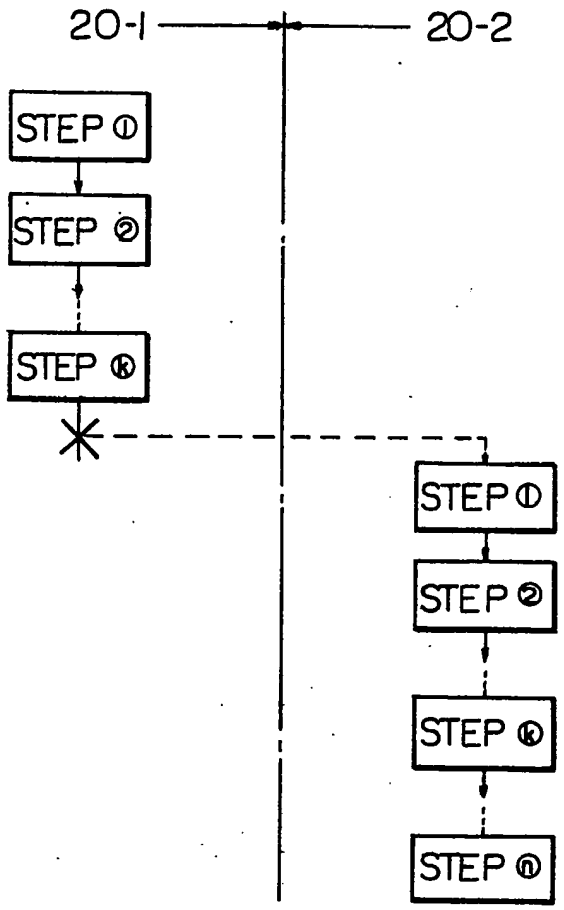

20-1
20-2
STEP ①
STEP ②
STEP ⓚ
STEP ①
STEP ②
STEP ⓚ
STEP ⓝ

Fig. 3

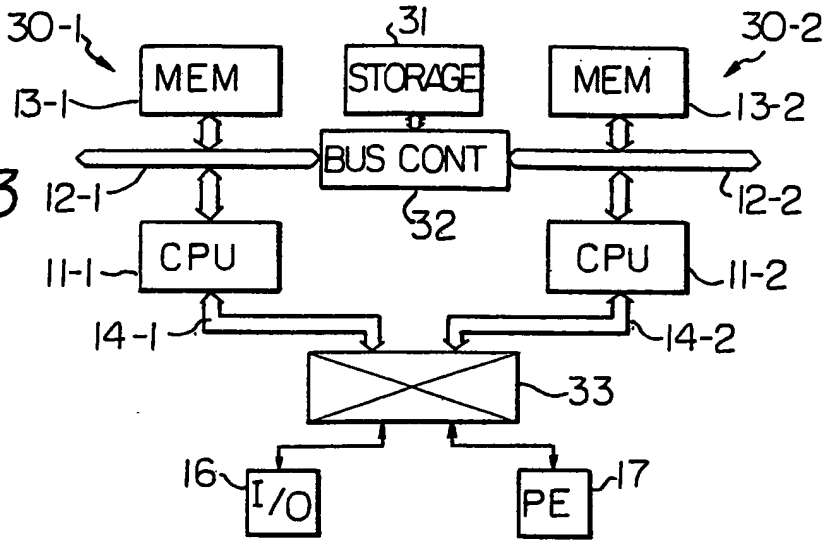

30-1
31
30-2
MEM
STORAGE
MEM
13-1
13-2
BUS CONT
12-1
12-2
32
CPU
CPU
11-1
11-2
14-1
14-2
33
16
I/O
PE
17

Fig. 4

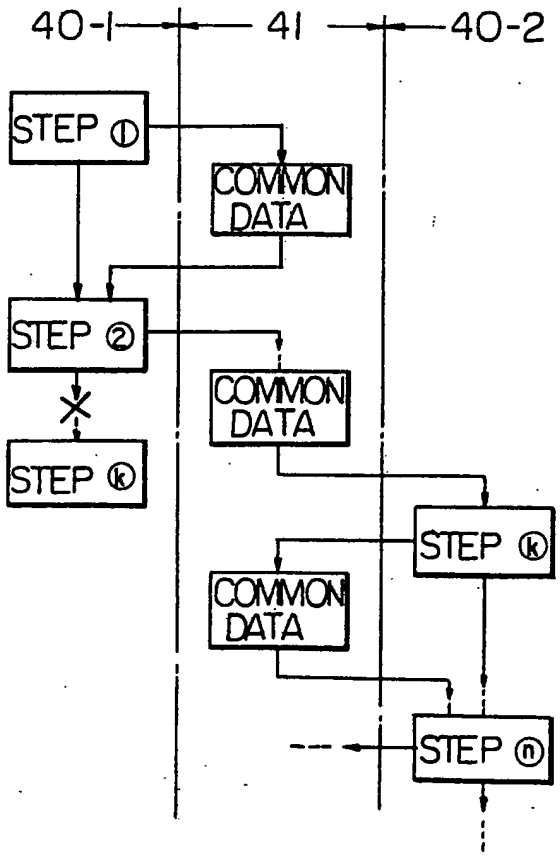

40-1
41
40-2
STEP ①
COMMON DATA
STEP ②
COMMON DATA
STEP ⓚ
STEP ⓚ
COMMON DATA
STEP ⓝ

Fig. 5

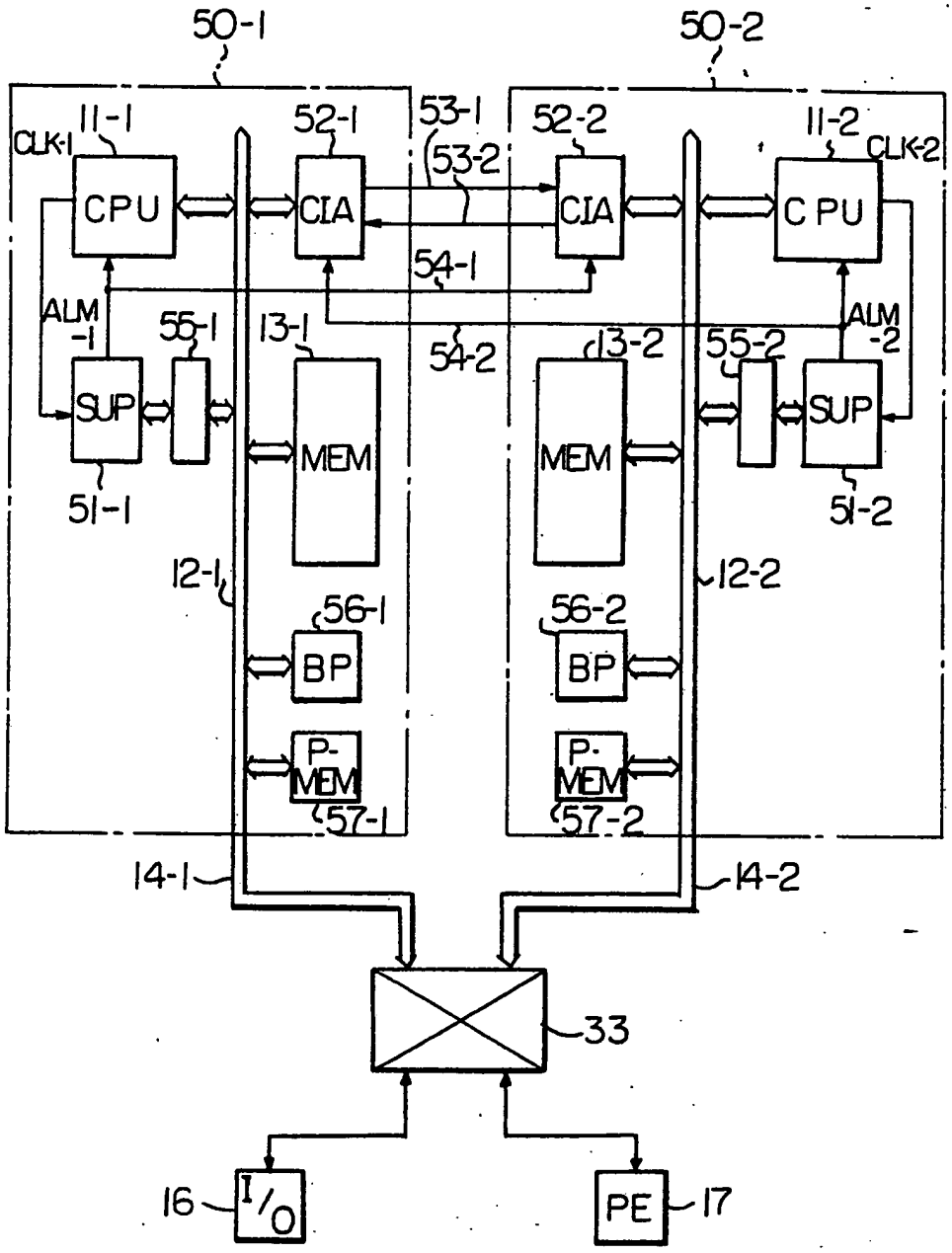

50-1
50-2
11-1
CLK-1
52-1
53-1
53-2
52-2
11-2
CLK-2
CPU
CIA
CIA
CPU
54-1
ALM-1
55-1
13-1
54-2
13-2
55-2
ALM-2
SUP
MEM
MEM
SUP
51-1
51-2
12-1
12-2
56-1
56-2
BP
BP
P-MEM
P-MEM
57-1
57-2
14-1
14-2
33
16
I/O
PE
17

Fig. 6

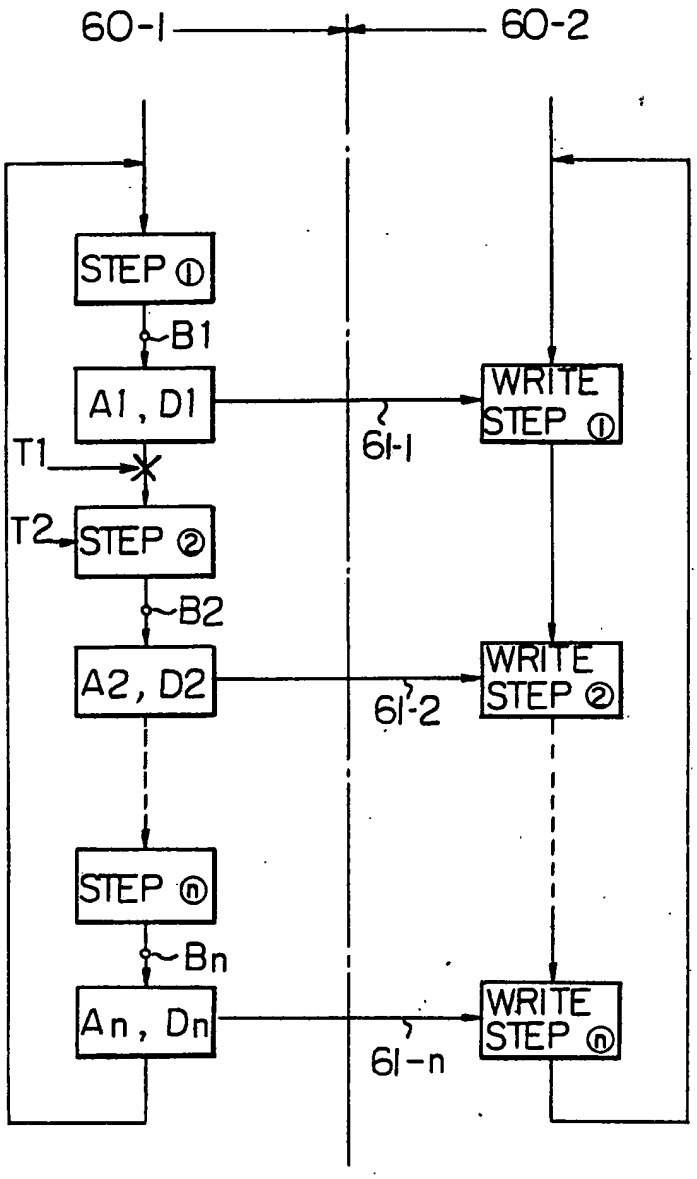

60-1
60-2
STEP ①
B1
A1, D1
WRITE STEP ①
61-1
T1
T2
STEP ②
B2
A2, D2
WRITE STEP ②
61-2
STEP ⓝ
Bn
An, Dn
WRITE STEP ⓝ
61-n

Fig. 7

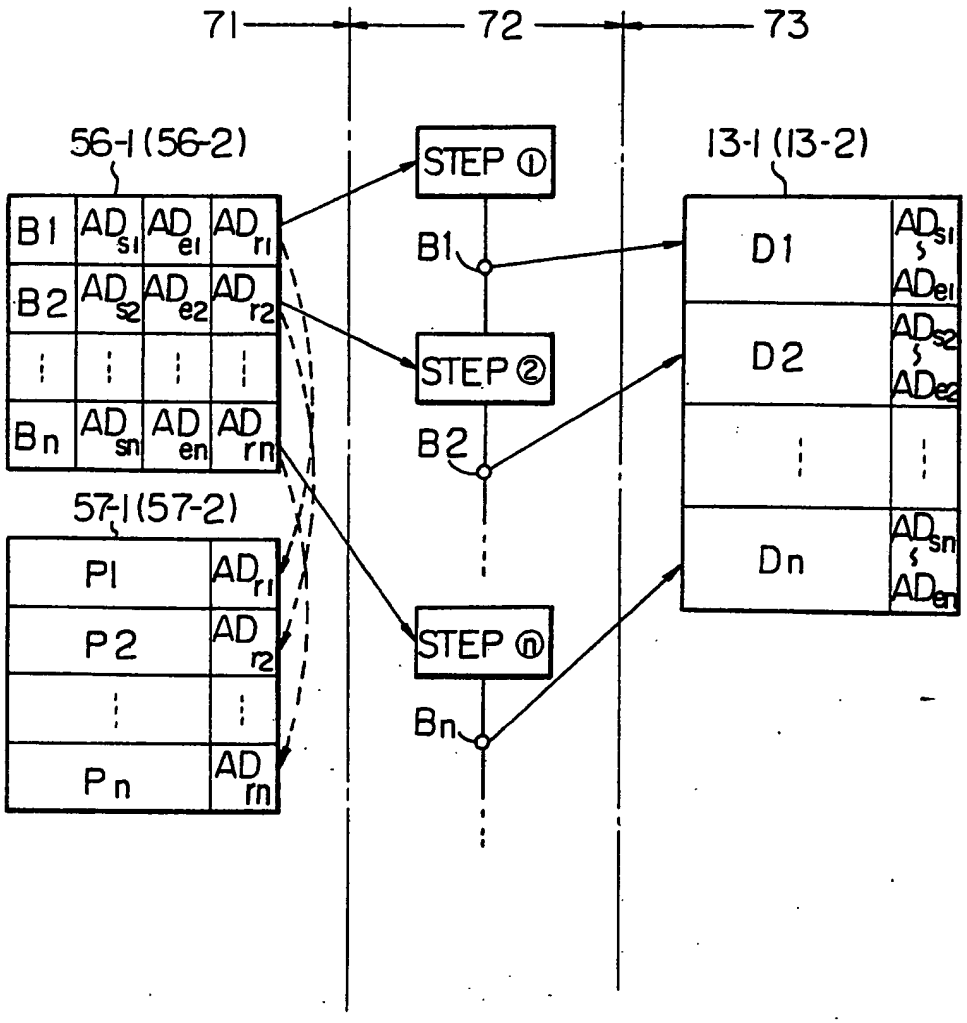

71
72
73
56-1(56-2)
B1 ADs1 ADe1 ADr1
B2 ADs2 ADe2 ADr2
Bn ADsn ADen ADrn
57-1(57-2)
P1 ADr1
P2 ADr2
Pn ADrn
STEP ①
STEP ②
STEP ⓝ
B1
B2
Bn
13-1(13-2)
D1 ADs1 ADe1
D2 ADs2 ADe2
Dn ADsn ADen

Fig. 8

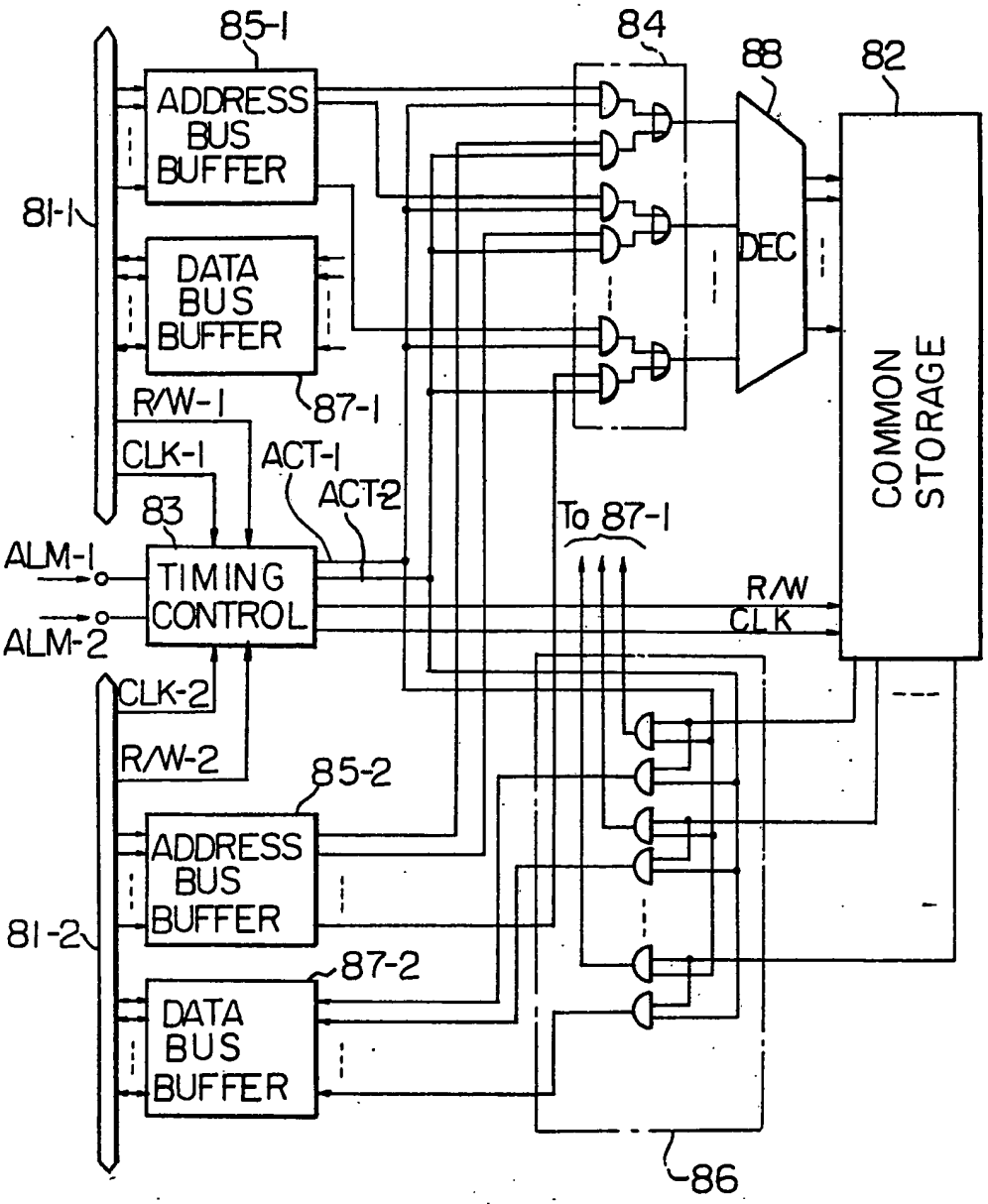
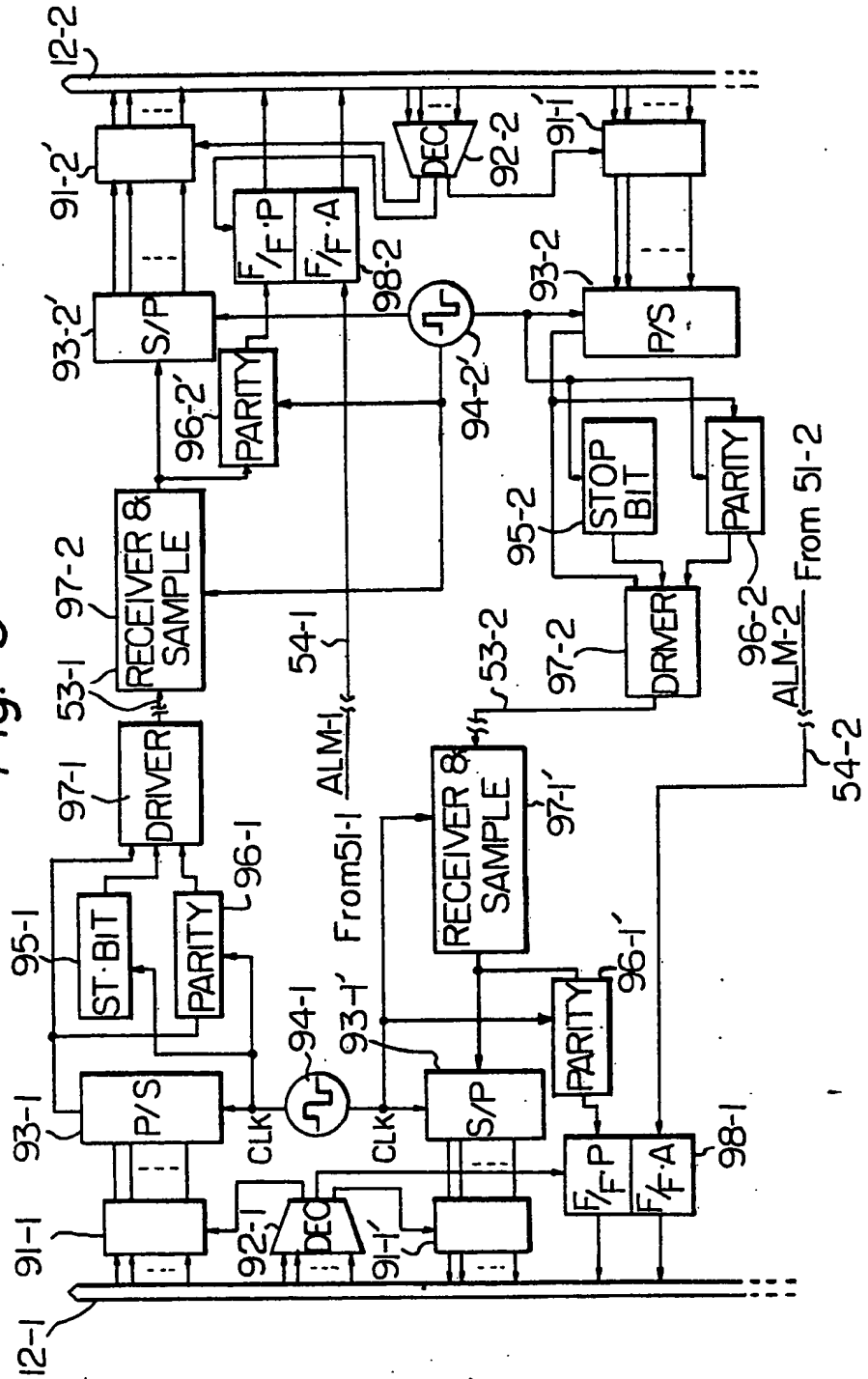

85-1
84
88
82
ADDRESS BUS BUFFER
81-1
DATA BUS BUFFER
DEC
COMMON STORAGE
87-1
R/W-1
CLK-1
ACT-1
ACT-2
83
To 87-1
ALM-1
TIMING CONTROL
R/W
CLK
ALM-2
CLK-2
R/W-2
85-2
ADDRESS BUS BUFFER
81-2
87-2
DATA BUS BUFFER
86

Fig. 9


12-2
91-2'
93-2'
S/P
96-2'
PARITY
97-2
RECEIVER & SAMPLE
53-1
97-1
DRIVER
95-1
ST. BIT
PARITY
96-1
93-1
P/S
91-1
12-1
CLK
94-1
92-1
DEC
91-1'
CLK
93-1'
S/P
From 51-1
ALM-1
54-1
RECEIVER & SAMPLE
97-1'
PARITY
96-1'
F/F·P
F/F·A
98-1
DEC
92-2
91-1'
F/F·P
F/F·A
98-2
93-2
P/S
94-2'
STOP BIT
95-2
PARITY
96-2
From 51-2
ALM-2
54-2
53-2
97-2
DRIVER

SPECIFICATION

Method for Carrying Out Synchronous Operation in a Multi-processor System

The present invention relates to a method for carrying out a synchronous operation in a multi-processor system.

Generally, a synchronous operation is performed in a multi-processor for the purpose of increasing the reliability of the data processing system, which multi-processor is comprised of at least a first processor and a second processor, where either one of the processors acts as a main processor and the other processor acts as an emergency processor. If an accident or trouble occurs in the main processor, the main processor is immediately switched to the emergency processor, so that the data processing system can still continue the normal data processing operation. In this case, the above mentioned synchronous operation must be performed in the first and second processors. This is because, if the synchronous operation is not performed therein, an undesirable blank is momentarily created in the normal data processing operation. When the processing is switched from the main processor to the emergency processor.

In the prior art, there are two typical methods for carrying out the synchronous operation. A first method is called an emergency-processor-stand-by method, and a second method is called a parallel-run method. In the first method, the emergency processor is always left in a stand-by condition, and if an accident or trouble occurs in the main processor, the main processor is switched to the emergency processor through a manual switching operation. Accordingly, in the first method, when the emergency processor is activated, it must always start the data processing at an initial program of the successive programs to be processed. Thus, the continuity of the data processing is lost. However, the first method has an advantage in that no special or complicated hardware is required, and only a simple switching means is necessary.

In the second method, both the first and second processors are activated simultaneously and carry out the same data processing in parallel. Therefore, when an accident or trouble occurs in one of the processors, the other processor can still continue the normal data processing. Accordingly, this second method is the best of the two, from a point of view that the continuity of the data processing cannot be lost. However, the second method has such disadvantages that, firstly, special and complicated hardware is required for controlling a common storage shared by the processors and for achieving synchronization between the processors, and secondly, the formation of the programs to be processed for the job in the data processing system becomes complicated.

Therefore, it is an object of the present invention to provide a method for carrying out a synchronous operation in a multi-processor system, which method is useful for maintaining the continuity of the data processing, as with the aforesaid second method, and for reducing the size of the hardware and the program, as with the aforesaid first method.

The present invention will be more apparent from the ensuing description, with reference to the accompanying drawings, wherein:

Fig. 1 is a block diagram of a multi-processor system used in the carrying out of a first method of the prior art for a synchronous operation;

Fig. 2 is a flow chart of successive processing steps processed in the multi-processor system illustrated in Fig. 1;

Fig. 3 is a block diagram of a multi-processor system used in the carrying out of a second method of the prior art for a synchronous operation;

Fig. 4 is a flow chart of successive processing steps processed in the multi-processor system illustrated in Fig. 3;

Fig. 5 is a block diagram of a multi-processor system used in the carrying out of the method of synchronous operation according to the present invention;

Fig. 6 is a flow chart of successive steps performed in the multi-processor system illustrated in Fig. 5;

Fig. 7 is an explanatory drawing used for explaining details of the method according to the present invention;

Fig. 8 is a block diagram of an interface circuit used in carrying out of the above mentioned second method of the prior art, and;

Fig. 9 is a block diagram of an interface circuit, that is CIA_s 52—1 and 52—2, according to the present invention.

In Fig. 1, which is a block diagram of a multi-processor system to which the above mentioned first method of the prior art is applied, the reference numerals 10—1 and 10—2 represent a first processor and a second processor, respectively. The first and second processors 10—1 and 10—2 contain a first CPU (Central Processing Unit) 11—1 and a second CPU 11—2, respectively, and the CPU_s 11—1 and 11—2 cooperate with respective first and second local memories (MEM) 13—1 and 13—2, via respective first and second common buses 12—1 and 12—2. If the first processor 10—1 acts as a main processor, the second processor 10—2 acts as an emergency processor and vice versa. The first processor 10—1, when acting as the main processor executes data communicating operation with an I/O (Input/Output) device 16 and peripheral equipment (PE) 17, via a common bus 14—1 and a switching means 15. If an accident or trouble occurs in the first processor 10—1, an operator manually switches the switching means 15 from one to the other status. This causes the second processor 10—2, which has been in a stand-by condition, to be activated. Thereafter, the second processor 10—2 executes the data communication operation with the I/O device 16 and the peripheral equipment 17, via a

common bus 14—2 and the switching means 15. In this case, as previously mentioned, the second processor 10—2 starts the data processing at an initial program and, as a result, the continuity of the data processing is lost.

A loss of continuity will now be explained with reference to Fig. 2. In Fig. 2, the successive processing steps of the first processor 10—1 (Fig. 1) are indicated in a left column 20—1, while the successive processing steps of the second processor 10—2 (Fig. 1) are indicated in a right column 20—2. When the first processor 10—1 (Fig. 1) processes processing steps ①, ② . . . ⓚ and an accident or trouble denoted by a symbol X) occurs after the step ⓚ, the second processor 10—2 (Fig. 1) starts processing the identical processing steps ①, ② . . . ⓚ and continues the processing until the step ⓝ . As previously mentioned, this first method of the prior art has the advantage that no special or complicated hardware is required, and only a simple switching means 15 is necessary.

In Fig. 3, which is a block diagram of a multi-processor system to which the above mentioned second method of the prior art is applied, for carrying out the synchronous operation between a first processor 30—1 and a second processor 30—2. The members in Fig. 3 denoted by the same reference numerals or symbols as used in Fig. 1, are identical to the members indicated in Fig. 1. Accordingly, a common storage 31, a bus controller (BUS CONT) 32 and a multiplexer 33 are distinctive members when compared to the members indicated in Fig. 1. As previously mentioned, the first and second processors are activated simultaneously, and carry out the same data processing in parallel. The first processor 30—1, when acting as the main processor, accomplishes the data communication together with the I/O device 16 and the peripheral equipment 17, via the multiplexer 33. The multiplexer 33 provides only the data supplied from the main processor 30—1. The common storage 31 is employed for always storing data which will be necessary for the second processor 30—2 to take over the job from the first processor when an accident or trouble occurs in the latter processor. The bus controller 32 is employed for sychronizing the timing clock signals used in the first and second processors, because these processors operate under respective basic clock pulse signals which are independent from each other.

The system illustrated in Fig. 3 has the aforesaid advantage that the continuity of the data processing is not lost. This continuity will now be explained with reference to Fig. 4. In Fig. 4, the successive processing steps of the first processor 30—1, the second processor 30—2 and the common storage 31 (Fig. 3) are indicated in a left column 40—1, a right column 40—2 and a middle column 41, respectively. When the first processor 30—1 (Fig. 3) processes processing steps ①, ② . . ., and an accident or a trouble (X) occurs before the steps ⓚ the second processor 40—2 processes the identical step ⓚ in place of the first processor 40—1 by utilizing the taking over data (COMMON DATA) stored in the common storage 31 (Fig. 3). Thus, this second method is more advantageous than the above mentioned first method from the point of view of continuity of the data processing. However, special and complicated hardware, such as the common storage 31, the bus controller 32 and the multiplexer, are required to carry out this second method. Further the formation of programs to be processed in the data processing system becomes complicated. A multi-processor system similar to the system illustrated in Fig. 3 is described in, for example, U.S. Patent No. 3,735,360, issued May 25, 1973.

A method for carrying out the synchronous operation, according to the present invention, provides the aforesaid advantages; that is, firstly, simplification of the formation of programs and hardware from those of the second method, and secondly, improvement of the continuity of the data processing over that of the first method. Fig. 5 is a block diagram of a multi-processor system used for carrying out the method of synchronous operation according to the present invention. In Fig. 5, the reference numerals 50—1 and 50—2 denote a first processor and a second processor, respectively. Either one of the processors accomplishes a data communication together with the I/O device 16 and the peripheral equipment 17, as does in the systems illustrated in Fig. 1 and Fig. 3, via the multiplexer 33. Each processor contains, the CPU 11—1 (11—2), the common bus 12—1) (12—2) and a supervisor circuit 51—1 (51—2). Accordingly, distinctive members in the present invention are a CIA (Communication Interface Adapter) 52—1 in the first processor 50—1 and a CIA 52—2 in the second processor 50—2. CIA_s 52—1 and 52—2 are connected by means of single communication wires 53—1 and 53—2. The supervisors 51—1 and 51—2 are also connected by means of single communication wires 54—1 and 54—2. Each of the supervisors 51—1 and 51—2 supervises an occurrence of an accident or trouble in the respective processors 50—1 and 50—2. Each supervisor may be comprised of, for example, a so-called "watch dog", timer and detects a stop of a basic clock pulse signal CLK—1 (CLK—2) and a runaway of software by cyclic refreshing the count memory in a supervisor register 55—1 (55—2). If the above mentioned stop of the basic clock pulse signal or the runaway of the software occurs in the processor, the supervisor determines that some accident or trouble has occurred in the processor, and produces an alarm signal ALM—1 (ALM—2). The alarm signal ALM—1 (ALM—2) is supplied to the CPU 11—1 (11—2) and, also, the CIA 11—2 (11—1), via the single communication wire 54—1 (54—2).

The method according to the present invention will now be explained with reference to Fig. 5 and Fig. 6. In Fig. 6, which is a flow chart of successive processing steps processed in the

multi-processor system of Fig. 5, the processing steps of the first processor 50—1, which is acting as a main processor, are revealed in a left column 60—1, while the processing steps of the second processor 50—2, which is acting as an emergency processor, are revealed in a right column 60—2. It is a remarkable feature of the method according to the present invention that the successive processing steps are segmented, in advance, into a plurality of processing steps at every breakpoint of the job. In the column 60—1, the segmented processing steps are indicated as step ①, step ② ... step ⓝ, and the break points are indicated by the symbols B1, B2 ... Bn. The above mentioned segmentation may freely be determined by a programmer. Therefore, the breakpoint (B1, B2 ... Bn) may be located at a segment, for example, a segment of "completion of receiving a message" or a segment of "completion of creating a data" and the like. With regard to Fig. 6, attention should be paid to the fact that steps (A1, D1), (A2, D2) ... (An, Dn) are inserted in the flow of the processing steps. The symbols D1, D2 ... Dn denote the resultant data which have been processed in the processing steps ①, ② ... ⓝ, respectively. The symbols A1, A2 ... An denote specified addresses at which the data D1, D2 ... Dn should be stored in the local memory 13—2 of the emergency processor 50—2, respectively. Accordingly, in Fig. 6, lines 61—1, 61—2 ... 61—n, respectively represent transfers of (address A1, data D1), (address A2, data D2) ... (address An, data Dn) to the emergency processor 50—2. Further, write steps ①, ② ... ⓝ indicate the write operation of the data D1, D2 ... Dn into the memory 13—2, at the specified addresses A1, A2 ... An, respectively. Since the specified addresses A1, A2 ... An are commony used in not only the memory 13—2 but, also, the memory 13—1, the memory 13—2 of the emergency processor 50—2 can always store a true copy of the data stored in the memory 13—1 of the main processor 50—1. Consequently, if the data to be processed in any of the processing steps ①, ② ... ⓝ, is updated, the corresponding data stored in the memory 13—2 is also updated at the same time.

As will be understood from the above explanation, in the method of the present invention, receiving data in the emergency processor are not all the data processed in each step, but merely the resultant data from the processing in each step. Since merely a small amount of the resultant data is intermittently supplied to the emergency processor 50—2, it is not necessary for the CIA 52—1 (52—2) to operate at very high operating speed, and also, it is not necessary for the CIA to be operated under control of a signal which indicates timing difference of steps between the processors 50—1 and 50—2. In other words, the CIA 52—1 (52—2) can be constructed as a simple circuit which is asynchronous with respect to the processors 50—1 and 50—2. Consequently, the size of hardware of the CIA_s is very small, because the essential member of the CIA_s may be made of conventional and simple parallel/serial converters. As a result, as previously mentioned, the connection between the processors 50—1 and 50—2, via the CIA_s 52—1 and 52—2, is not made of data buses but, simple, the single communication wires 53—1 and 53—2. Contrary to this, in the aforesaid second method of the prior art (refer to Fig. 3), an interface circuit can not be made by using a simple circuit, such as the CIA_s 52—1 and 52—2, and accordingly, the interface circuit becomes complicated in construction, for example, as illustrated in Fig. 8. Further, a formation of programs is very simple, when compared to the second method of the prior art. This is because, as mentioned above, the CIA_s 52—1 and 52—2 can be constructed as a so-called asynchronous circuit.

When the supervisor 51—1 detects an accident or trouble in the processor 50—1 at a time T1 (see symbol X at the time T1 in Fig. 6), the supervisor 51—1 produces the alarm signal ALM—1 and supplies it to the CPU 11—1 and the CIA 52—2 via the wire 54—1. The CPU 11—2 always supervises the progress of the processing steps, and accordingly, the CPU 11—2 can discriminate the last processing step which has been processed in the processor 50—1 (refer to the left column 60—1). And the CPU 11—2 starts processing from a specified program, which program is specified, among the successive programs, by a restart address which is allotted to the corresponding breakpoint B1, because the accident or trouble now occurs at the time T1. The restart addresses allotted to all the breakpoints B1, B2 ... Bn, are also shared in advance, to the processor 50—2. The processor 50—2 starts processing the program specified by the restart address, by using the last updated data which was supplied through the line 61—1, that is, the transfer of (address A1, data D1), and is stored in the memory 13—2.

If the supervisor 51—1 detects an accident or trouble in the processor 50—1 at a time T2, the CPU 11—2 also starts processing from the program which is specified by the restart address corresponding to, not the breakpoint B2, but the breakpoint B1. This is because, since the processing step ② is not yet completed at the time T2, the resultant data of this step ② is not yet updated in the memory 13—2. Accordingly, the CPU 11—2 can not use the resultant data of the step ②. Thus, the CPU 11—2 returns and starts processing the program which corresponds to the breakpoint B1.

The details of carrying out the above mentioned method of the present invention will now be explained with reference to Fig. 7. In Fig. 7, a flow of steps indicated in a middle column 72 is identical with the column 60—1 of Fig. 6. A left column 71 has a breakpoint table 56—1 (56—2) and a program memory 57—1 (57—2). The breakpoint table (BPT) 56—1 (56—2) can be seen in Fig. 5, and the program memory (P.MEM) 57—1 (57—2) can also be seen in Fig. 5. These

members 56—1 (56—2) and 57—1 (57—2) are comprised of, for example, ROM_s (Read Only Memories). A right column 73 has the local memory 13—1 (13—2). The breakpoint table is segmented by the breakpoints B1, B2 ... Bn. The breakpoint table stores, firstly, the starting addresses AD_{s1}, AD_{s2} ... AD_{sn} at respective breakpoints B1, B2 ... Bn, secondly, the ending addresses AD_{e1}, AD_{e2} ... AD_{en} at respective breakpoints B1, B2 ... Bn and, thirdly, the restart addresses AD_{r1}, Ad_{r2} ... AD_m at respective breakpoints B1, B2 ... Bn. The program memory stores successive programs P1, P2 ... Pn at the address corresponding to the restart addresses AD_{r1}, AD_{r2} ... AD_m, respectively. Then, the memory stores the resultant data of respective steps ①, ② ... ⓝ, at the respective memory are as defined by the addresses (AD_{s1} AD_{e1}) (AD_{s2} AD_{e2}) ... (AD_{sn} AD_{en}) respectively. If an accident or trouble occurs in the processor 50—1, the processor 50—2 starts processing from the program P1 specified by the restart address AD_{r1}, when, for example, the accident or trouble occurs at the time T1.

As will be understood from the above explanation the method of the present invention has a disadvantage. This disadvantage resides in the fact that, when a switch from the processor 50—1 to the processor 50—2 takes place due to the occurrence of an accident or trouble, the processor 50—2 must repeat the same data processing operation which has already been carried out in the processor 50—1. For example, in Fig. 6, when the accident or trouble occurs at the time T2, the processor 50—2 must repeat the operation of the data processing with respect to the same step ②, although a large part of the same data processing with respect to this step ② has been completed in the processor 50—1. However, it should be noted that the continuity of the data processing in any of the two successive processing steps, is not lost. Further, the above mentioned disadvantage of the present invention may sufficiently be cancelled by the advantages of the present invention. The advantages are, as previously mentioned, a simplification of hardware, especially the interface circuit represented by CIA_s 52—1 and 52—2, and a simplification of the formation of the programs.

The simplification of the hardware of the interface circuit will now be explained with reference to Figs. 8 and 9. Fig. 8 is a block diagram of an interface circuit employed in the second method of the prior art. In Fig. 8, the reference numerals 81—1 and 81—2 denote common buses similar to the common buses 12—1 and 12—2 in Fig. 3. The reference numeral 82 denotes a common storage similar to the common storage 31 in Fig. 3. All the remaining members correspond to the bus controller 32 in Fig. 3. A switch timing control circuit 83 receives the alarm signals ALM—1, ALM—2, the basic clock pulse signals CLK—1, CLK—2, and the read/write indicating signals R/W—1, R/W—2. Then, the circuit 83 produces a first activation signal ACT—1 and a second activation signal ACT—2, and also, produces the signals R/W—1, and CLK—1, when the first processor acts as a main processor. In this case, the signal ACT—1 is logic "1" and the signal ACT—2 is logic "0". The address selector 84 selects the address data from an address bus buffer 85—1, because the signal ACT—1 is now logic "1". Also the data selector 86 provides the data from the common storage 82 to a data bus buffer 87—1, because the signal ACT—1 is now logic "1". The data which is read from the storage 82 or is written thereinto, is specified by the address data applied to the steps 82 via an address decoder 88. As seen from Fig. 8, the interface circuit employed for carrying out the second method requires large size hardware, such as address bus buffers 85—1, 85—2, data bus buffers 87—1, 87—2, the address decoder 88 and the large capacity storage 82.

Contrary to the above, the interface circuit of the present invention, that is the CIA_s 52—1 and 52—2, requires no large size hardware, as can be seen in Fig. 9. In Fig. 9, the members indicated by the same reference numerals or symbols as those used in Fig. 5 represent the identical as those indicated in Fig. 5. Since the first processor is a main processor, the following explanation will be directed to only the first processor side. A register 91—1 is an entre of the CIA 52—1. A decoder (DEC) 92—1 decodes the address data transferred on the common bus 12—1 and specifies the register 91—1. The register 91—1 momentarily stores each parallel data of (A1, D1), (A2, D2) ... (An, Dn) transferred on the bus 12—1. The data from the register 91—1 is applied to a parallel/serial converter (P/S) 93—1. The converter 93—1 produces the serial bits of data in keeping time with clock pulse (CLK) supplied from a clock generator 94—1. The clock pulse CLK is also applied to both a stop-bit generator (ST BIT) 95—1 and a parity-bit generator (PARITY) 96—1. The data from the converter 93—1, together with both the stop bit from the generator 95—1 and the parity bit from the generator 96—1, is transmitted to the second processor, via a transmitting driver 97—1 and the single communication wire 53—1. Thus, the data (A1, D1), (A2, D2) ... (An, Dn) (see Fig. 6) are sequentially supplied to the memory 13—2 of Fig. 5. A receiving data sampler (SAMPLE) 97—1', a parity checker (PARITY) 96—1', a serial/parallel converter (S/P) 93—1' and a register 91—1' are all activated when the first processor acts as an emergency processor.

When the supervisor 51—1 detects an accident or trouble occurring in the first processor, the alarm, signal ALM—1 is applied to a status indicator 98—2, which is comprised of a flip flip F/F . A and a flip flop F/F . P. The signal ALM—1 sets the flip flop F/F . A to alarm status. When an address decoder 92—2 specifies the indicator 98—2, the second processor determines that the first processor is not in a normal condition. Then, the second processor finds out the aforementioned last break-point (B1,

B2 . . . Bn) and starts the processing operation in the manner as already explained with reference to Figs. 6 and 7. The flip flop F/F . P indicates whether or not the parity check is right.

It should be noted that, in Fig. 9, all the circuit elements are of small size and simple construction. Furthermore, there is no buses as illustrated in Fig. 8, but single wires. In addition, the size of the decoder 92—1 (92—2) is very small when compared to the address decoder 88 of Fig. 8.

As mentioned above, the method for carrying out the synchronous operation in the multi-processor system according to the present invention, is very useful for maintaining the continuity of the successive data processing steps, without introducing any large size and complicated hardware.

Claims

1. A method for carrying out synchronous operation in a multi-processor system which is comprised of at least a first processor and a second processor, the first processor executes processing of data along the sequence of the successive processing steps, in accordance with programs, which method comprises the steps of:

(a) transferring the resultant data from the first processor to the second processor every time a processing step is completed;

(b) storing each resultant data in a memory of the second processor, and;

(c) causing the second processor to take over the data processing when an accident or trouble occurs in the first processor, and to start processing from one of the programs which exist nearest the occurrence of the accident or trouble, by utilizing the last updated resultant data stored in the memory.

2. A method as set forth in claim 1, wherein the successive processing steps are segmented, in advance, by breakpoints of a job to be processed.

3. A method as set forth in claim 2, wherein a restart address is alloted, in advance, to each break point, and also to each program, and the second processor picks up the restart address alloted to the breakpoint which exists nearest the occurrence of the accident or trouble, and then, the second processor starts processing from the program which is specified by the same restart address.

4. A method as set forth in claim 1, wherein the step (a) is performed by means of a communication interface adapter, which is comprised of a parallel/serial converter and contained in the first processor.

5. A method as set forth in claim 4, wherein a serial/parallel converter is contained in the second converter, the parallel/serial converter and the serial/parallel converter being connected by means of single communication wires.

Printed for Her Majesty's Stationery Office by the Courier Press, Leamington Spa, 1980. Published by the Patent Office, 25 Southampton Buildings, London, WC2A 1AY, from which copies may be obtained.